A Momentum-Based Swing Trading Algorithm with Supplemental Confirmation Signals

Richard Wang

Introduction:

The goal of this project was to explore the capabilities of different trading algorithms operating on the U.S. stock market. In particular interest was expanding upon the classical concept of swing trading. Traditional strategies hinge upon capitalizing on market price fluctuations to secure profits, often holding positions ranging from a few days to weeks.[1] While swing trading can have a lot to offer in terms of viability and profit-making, it simultaneously can be prone to many risks. One such weakness is its susceptibility to false signals.[2] Indeed, factors spanning from market noise to even weak trend confirmation create deceptive entry signals that can lead to unfavorable losing positions.

To combat these issues, my approach to swing trading includes a momentum-based strategy consisting of SMA (Simple Moving Average) crossover and RSI (Relative Strength Index) filter. Furthermore, I am incorporating volatility contexts through Bollinger Bands and a volume metric as confirmation signals to reduce weak and false entry signals. Through this, I hope to produce increased profit returns and create stronger signals for market entry.

Methodology:

This strategy focuses on trading medium-sized market cap stocks in the U.S. The particular stock of interest is Box, Inc. (BOX), a cloud storage company based in California.

- **Data Collection**

  Market data for BOX (OHLCV) was collected through yfinance, a Python library to fetch financial data from Yahoo Finance.[3] Due to the nature of swing trading, I opted for daily data.

[1] *Swing trading: Strategies and Insights for Successful Trading*. Swing trading: A complete guide for investors | TD Direct Investing. (n.d.). https://www.td.com/ca/en/investing/direct-investing/articles/swing-trading

[2] Edu, Ai. (2025, May 23). *Mastering the Art of Swing Trading with Technical Indicators*. https://www.ainvest.com/news/mastering-art-swing-trading-technical-indicators-2505/

[3] *Yfinance*. PyPI. (n.d.). https://pypi.org/project/yfinance/

- **Signal Generation Logic**

  I focused on two signals – an entry (BUY) signal and an exit/close (SELL) signal.

  For a **BUY** signal: a weighted signal score was used to assess the likelihood of market entry. The score was calculated based on SMA crossover, RSI filter, Bollinger Bands, and Volume spikes. For SMA crossover, hyperparameters for the fast period and slow period were used for optimization and better data-fitting. RSI, Bollinger Bands, and Volume were incorporated using a percent change from their expected values. The specific weightings used along with the minimum threshold for a valid signal were all strategy hyperparameters fitted for better performance. The purpose of using weighted signaling was to better respond to dynamic and volatile markets.

  For a **SELL** signal: 2 conditions were used to generate a sell signal. The first was a stop loss set at 3% of the entry price. This was to prevent overly massive losses. The second condition consisted similarly of the previously used factors: SMA crossover, RSI, and Bollinger Bands. If any one of these reached the threshold for market exit, a sell signal was consequently generated.

- **Position Sizing**

  Position sizing was calculated based on the risk for each trade. Set at 1%, the risk/loss per trade and the stop loss percentage (3%) were both used to calculate the number of shares to be traded. This approach helped to balance profit and loss.

- **Backtesting Procedure**

  Backtesting was done through backtrader, a Python library used for testing strategies through a comprehensive framework.[4] Set over a one-year period from 2020-21, the strategy was tested on daily market data from BOX. The initial portfolio value was set to $10,000. The final portfolio value was calculated, along with profits/losses and other relevant metrics such as the annualized Sharpe ratio.

- **Overall Strategy Implementation**

  The strategy was created in Python through backtrader. Utilizing their indicators class, hyperparameter tuning and signal generation logic were implemented as described above. Order logs and notifications were also implemented to provide detailed reports of order creation/execution. Detailed code along with all data sources and libraries are provided in the appendix section.

[4] Rodriguez, D. (n.d.). *Welcome to Backtrader*. Backtrader. https://www.backtrader.com/

Results:

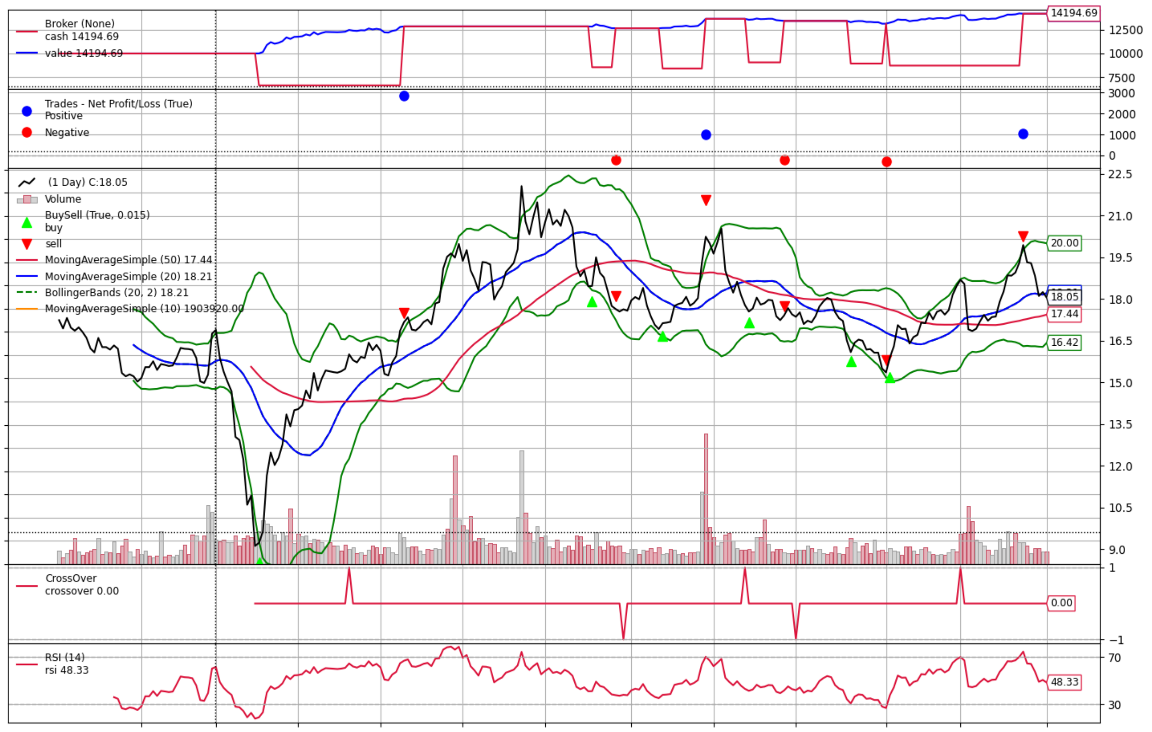

Figure 1.

The graph to the left represents backtesting of the swing trading algorithm **with** confirmation signals. Key evaluation metrics are shown below:

Starting Portfolio Value: $10,000

Final Portfolio Value: $14,194.69

PnL: $4,194.69

Sharpe Ratio (Annualized): 2.62

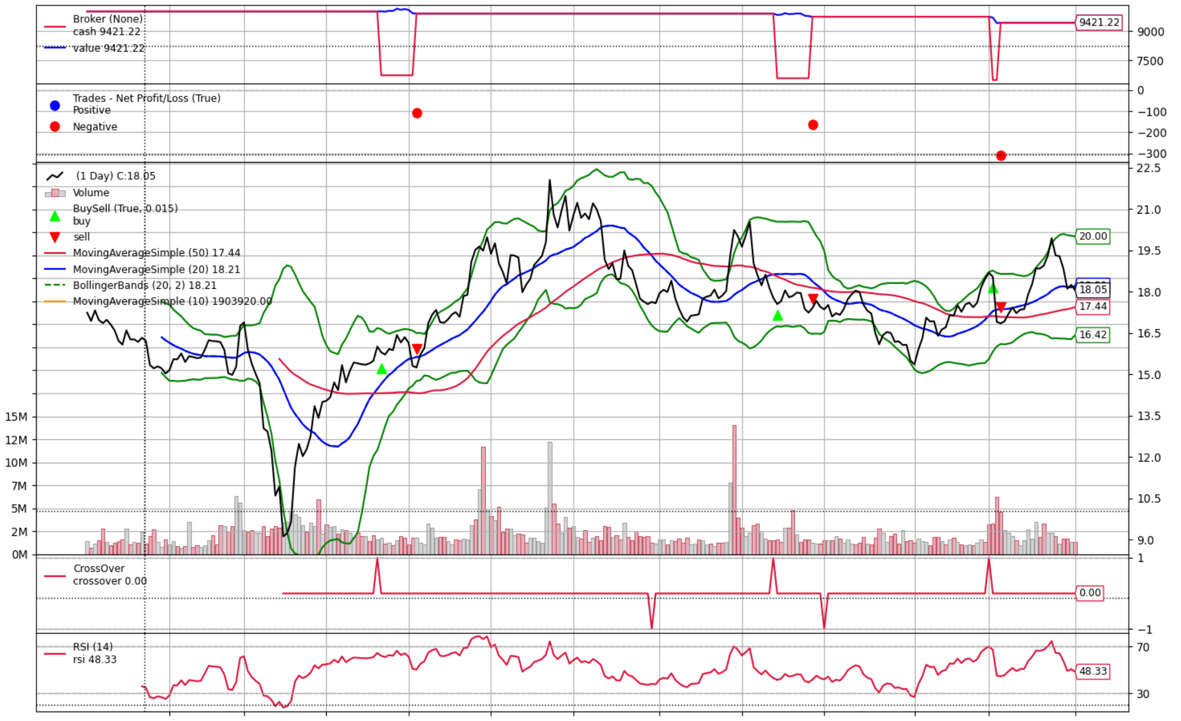

Figure 2

The graph to the left represents similar backtesting of the swing trading algorithm **without** confirmation signals (purely SMA crossover). Key evaluation metrics are below:

Starting Portfolio Value: $10,000

Final Portfolio Value: $9,421.22

PnL: $-578.78

Sharpe Ratio (Annualized): -1.70

Discussion:

The swing trading algorithm worked well when used in tandem with the supplemental confirmation signals including an RSI filter, Bollinger Bands, and volume metrics. Over a one-year period, the strategy made around 42% profit returns and beat the overall market. The annualized Sharpe ratio was 2.62, confirming good performance.

In contrast, the algorithm with solely SMA crossover performed poorly over the one-year period. It achieved negative profits and had an annualized Sharpe ratio of -1.70.

What worked well with the strategy were the confirmation signals. These helped to generate stronger, more robust entry signals that were missed by solely SMA crossover. Furthermore, risk position sizing helped to maximize profits per trade while simultaneously minimizing potential losses. Finally stop losses were great in keeping losses at a minimum, while also providing capitalization on the volatile market.

Conclusion:

While swing trading is a viable strategy when operating on volatile stock markets, it can be prone to false signals giving rise to unfavorable positions. As a result, there is a benefit in incorporating additional confirmation signals to validate market entries and exits. Through backtesting and performance evaluation, promising confirmation signals for medium-sized market cap U.S. stocks like BOX include RSI, Bollinger Bands, and volume metrics. These factors provide additional information regarding volatility, over/under buying, and general market trends that can prevent weak and even false signals.

As an initial development, this swing trading strategy can be expanded upon in future works. Particularly, instead of maintaining just one singular position at any given time, having the option to enter multiple different market positions adds greater depth and can enhance performance. Additionally, operating on a multi-stock portfolio will increase diversity and lead to more nuanced strategies that can further extend the one presented here. Ultimately, while this swing trading strategy shows promise in market performance, adding greater complexity may provide new insights into the very nature of swing trading itself.

Citations

BDPO.io. (2023, March 8). *Creating a Profitable Trading Strategy using RSI and Bollinger Bands*. Medium. https://medium.com/@BDPO/creating-a-profitable-trading-strategy-using-rsi-and-bollinger-bands-aacaa89a571b

Datetime - Basic Date and Time Types. Python documentation. (n.d.). https://docs.python.org/3/library/datetime.html

Davda, J. (2022, July 16). *Backtrader for Backtesting (Python) - A Complete Guide*. Quantitative Trading Ideas and Guides - AlgoTrading101 Blog. https://algotrading101.com/learn/backtrader-for-backtesting/

Edu, Ai. (2025, May 23). *Mastering the Art of Swing Trading with Technical Indicators*. https://www.ainvest.com/news/mastering-art-swing-trading-technical-indicators-2505/

Pandas. pandas. (n.d.). https://pandas.pydata.org/

Rodriguez, D. (n.d.). *Welcome to Backtrader*. Backtrader. https://www.backtrader.com/

Swing trading: Strategies and Insights for Successful Trading. Swing trading: A complete guide for investors | TD Direct Investing. (n.d.). https://www.td.com/ca/en/investing/direct-investing/articles/swing-trading

Visualization with Python. Matplotlib. (n.d.). https://matplotlib.org/

Yfinance. PyPI. (n.d.). https://pypi.org/project/yfinance/

Appendix Section

Strategy Implementation and Backtesting Code:

```python
# Import necessary libraries
import yfinance as yf
import pandas as pd
import backtrader as bt
from backtrader.indicators import RSI, BollingerBands, MovingAverageSimple, CrossOver
import matplotlib.pyplot as plt
import datetime
```

```python
class SwingTrade(bt.Strategy):
    # Relevant Parameters for our strategy
    params = (
        ('rsi_period', 14),
        ('overbought', 70),
        ('rsi_neutral', 50),
        ('bb_period', 20),
        ('bb_std', 2),
        ('pfast', 20),
        ('pslow', 50)
    )



    # Log our trades
    def log(self, txt, dt=None):
        dt = dt or self.datas[0].datetime.date(0)
        print(f'{dt.isoformat()} {txt}')



    # Initialization
    def __init__(self):
        self.dataclose = self.datas[0].close

        self.order = None

        self.entry_price = None

        self.slow_sma = MovingAverageSimple(self.datas[0],
                                            period = self.params.pslow)

        self.fast_sma = MovingAverageSimple(self.datas[0],
                                            period=self.params.pfast)

        self.crossover = CrossOver(self.fast_sma, self.slow_sma)

        self.rsi = RSI(period = self.params.rsi_period)

        self.bbands = BollingerBands(period = self.params.bb_period,
                                     devfactor = self.params.bb_std)

        self.avg_volume = MovingAverageSimple(self.datas[0].volume, period=10)


    # Streamline the Orders
    def notify_order(self, order):
        # Check if order is submitted/accepted
        if order.status in [order.Submitted, order.Accepted]:
            return

        # Check if order is completed
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(f'BUY EXECUTED, {order.executed.price:.2f}')
            elif order.issell():
                self.log(f'SELL EXECUTED, {order.executed.price:.2f}')

        elif order.status in [order.Canceled, order.Margin, order.Rejected]:
            self.log('Order Canceled/Margin/Rejected')

        # Reset order
        self.order = None



    # Main Logic for Signal Generation
    def next(self):
        # Check for any current open orders
        if self.order:
             return

        # Check if currently in the market
        if not self.position:
            # BUY if weighted signal score exceeds min threshold
            signal_score = 0

            # SMA crossover
            if self.crossover > 0:
                signal_score += 1

            # RSI
            if self.rsi[0] < self.params.rsi_neutral:
                pct_change = ((self.params.rsi_neutral -
                              self.rsi[0])/self.params.rsi_neutral)
                signal_score += pct_change

            # Bollinger Bands
            if (self.dataclose[0] <= self.bbands.lines.mid[0]):
                pct_change = (self.bbands.lines.mid[0] -
                              self.dataclose[0])/(self.bbands.lines.mid[0] -
                                        self.bbands.lines.bot[0])
                signal_score += pct_change

            # Volume Metric
            if (self.datas[0].volume > 1.5 * self.avg_volume):
                signal_score += 0.1

            # Generate BUY signal
            if signal_score >= 1.15:
                self.log(f'BUY CREATE {self.dataclose[0]:2f}')
                self.order = self.buy()
                self.entry_price = self.dataclose[0]

        # Signal to CLOSE trades
        else:
            # Stop loss signal
            stop_price = self.entry_price * (1-0.03)
            if (self.dataclose[0] <= stop_price):
                self.log(f'STOP LOSS CREATE {self.dataclose[0]:2f}')
                self.order = self.close()
            # Otherwise check for swing signals and relevant filters
            elif (self.crossover < 0 or
                    self.rsi[0] > self.params.overbought or
                    self.dataclose[0] >= self.bbands.lines.top[0]):
                self.log(f'CLOSE CREATE {self.dataclose[0]:2f}')
                self.order = self.close()

```

```python
# Position Sizing
class RiskSizer(bt.Sizer):
    params = (
        ('risk_per_trade', 0.01),
        ('stop_loss_pct', 0.03)
    )

    def _getsizing(self, comminfo, cash, data, isbuy):
        risk_amt = cash * self.params.risk_per_trade
        stop_loss_amt = data.close[0] * self.params.stop_loss_pct
        return int(risk_amt/stop_loss_amt)

```

```python
# Backtesting strategy
if __name__ == '__main__':

    # Load historical data for backtesting
    start_date = datetime.datetime(2020, 1, 1)
    end_date = datetime.datetime(2021, 1, 1)

    yfData = yf.download("BOX",
                         start=start_date, end=end_date, interval='1d')

    # Flatten data for backtesting with backtrader
    if isinstance(yfData.columns, pd.MultiIndex):
        yfData.columns = yfData.columns.get_level_values(0)

    data = bt.feeds.PandasData(dataname=yfData)

    # Create Cerebro Object
    cerebro = bt.Cerebro()

    # Add our Dataset
    cerebro.adddata(data)

    # Add our Swing Trade Algorithm
    cerebro.addstrategy(SwingTrade)

    # Custom Position Sizing
    cerebro.addsizer(RiskSizer)

    # Add Analyzer for Evaluation Metrics
    cerebro.addanalyzer(bt.analyzers.SharpeRatio_A, _name='sharpe_ratio',
                        timeframe=bt.TimeFrame.Days)

    # Starting Portfolio Value
    cerebro.broker.setcash(10000)

    start_portfolio_value = cerebro.broker.get_value()

    # Run the backtest
    results = cerebro.run()

    end_portfolio_value = cerebro.broker.get_value()
    pnl = end_portfolio_value - start_portfolio_value

    # Print Relevant Results/Metrics
    print(f'Starting Portfolio Value: {start_portfolio_value:2f}')
    print(f'Final Portfolio Value: {end_portfolio_value:2f}')
    print(f'PnL: {pnl:.2f}')
    print("Sharpe Ratio (Annualized):",
          results[0].analyzers.sharpe_ratio.get_analysis()['sharperatio'])

    # Plot Results
    cerebro.plot()
    plt.show()

```